# Apple Assembly Line

$1.80

Volume 7 -- Issue 5 February, 1987


In This Issue...



## Apple IIgs Technical Reference, by Michael Fischer

Hot off the press, over two pounds and nearly 700 pages of useful reference material for your IIgs. As far as I know, this is the first book to reveal the NUMBERS you need to use the tools in the toolbox. Published by Osborne/McGraw-Hill at $19.95, our price will be an even $19.

You know Michael Fischer as the author of 65816/65802 Assembly Language Programming, same publishers. That book is apparently doing well, because McGraw-Hill just raised the price from $18.95 to $21.95. Our price went up a little too....

## Product Information

Look inside for more detail on two more exciting new products: ProVIEW, by Doug McComsey is a professional tool for zapping your ProDOS disks; the new ProDOS-based S-C Disassembler fills the need for a fast, powerful, and flexible intelligent disassembler operating under ProDOS together with the S-C Macro Assembler.

For years we have been selling the little book by Caxton Foster, "Real Time Programming". It was a little gem, but we cannot get them anymore. The publisher told us they have no schedule for reprinting it. We also miss Foster's excellent book on cryptograms & computers. Both books used the Apple for examples.

Pre-Allocating DOS Files.........................Bill Morgan

Lately I've been working on a project that requires saving large files to disk as fast as possible. The information arrives in 256-byte chunks, with brief pauses between the chunks, so it's not too tough to write to a hard disk during the pauses. I found that things worked fine while saving to an existing file, but when I had to create a new file I got into trouble with the time DOS wastes allocating sectors one by one and constantly reading and re-writing the VTOC. Time for some speedup tricks!

I remembered that the S-C Word Processor uses a special technique to save its files, calling DOS routines to allocate enough sectors and create a track/sector list before actually writing the file. This approach requires that we know how big the file is before opening it, but that's OK because I do have that information. Another problem quickly showed up: the Word Processor knows that its files cannot be bigger than 120 sectors, and therefore will need only one sector's worth of t/s list. My files can be several hundred sectors long, so I have to be ready for anything.

The following machine language routines, along with the Applesoft driver, will create the track/sector lists for a file of any length.

This approach to creating a file does require some non-standard use of DOS routines, so a little patching is necessary. We will be using the File Manager's ALLOCATE.SECTOR routine, which does all that extra VTOC handling. Since we're going to repeatedly call that routine until we're done, it's a waste of time to shuffle the VTOC back and forth. Patching an RTS in at $AFFD stops the extra VTOC action. Since we're sort of sneaking into the File Manager's territory, we can't rely on the normal error handling. The patch at $B392 grabs control in the event of an error, so we can remove our patches and handle the error our way.

These programs will almost certainly fail under a modified DOS, so compare the code at the patch locations to standard DOS before trying it out.

Note: Some of the techniques here look odd even to me, but the original program had only a limited space for machine language routines and lots of room for a compiled Applesoft program. That's why, for example, the catalog is updated and rewritten from BASIC in lines 1150-1190, rather than from the machine language.

The Applesoft code is really pretty simple. We issue a DOS UNLOCK command to get the catalog sector containing our file into memory. (If the file doesn't already exist, then the ONERR code at line 10000 creates it.) Once we have the catalog information we can call the machine language routines to 'open' the file by locating the first track/sector list and then allocate the file. Then it's just a matter of looping along collecting data and writing it out to the next sector in the

S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners....................$20
ProDOS Upgrade Kit for Version 2.0 DOS owners.........................$30
Source Code of S-C Macro 2.0 (DOS only)..................additional $100
Source Code of S-C Macro 2.0 (ProDOS only)...............additional $100
S-C Disassembler (ProDOS only)...without source code $30, with source $50
ProVIEW (ProDOS only).................................................$20
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility......without source code $20, with source $50
RAK-Ware DISASM (DOS only).......without source code $30, with source $50
S-C Word Processor (with complete source code)........................$50
DP18 Source and Object................................................$50
Double Precision Floating Point for Applesoft (with source code)......$50
ES-CAPE (Extended S-C Applesoft Program Editor)..........
Including Version 2.0 With Source Code..........$50
ES-CAPE Version 2.0 and Source Code Update (for Registered Owners)....$30
Bag of Tricks 2 (Quality Software)........................($49.95)  $45 *
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00)  $50 *
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
(Available:  6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048,
8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)

AAL Quarterly Disks...........................each $15, or any four for $45
Each disk contains the source code from three issues of AAL:
Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec.
(All source code is formatted for S-C Macro Assembler.  Other assemblers
require some effort to convert file type and edit directives.)

Diskettes (with hub rings)........................ package of 25 for $15 *
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
(Cardboard folders designed to fit 6"X9" Envelopes.)    or $25 per 100 *
Envelopes for Diskette Mailers...........................  6 cents each

65802 Microprocessor (Western Design Center)...................($95)  $50 *
quikLoader EPROM System (SCRG).................................($179) $170 *
PROmGRAMER (SCRG)...........................................($149.50) $140 *
Minuteman 250 Uninterruptible Power Supply...................($359) $320 +

"Apple //gs Technical Reference", Fischer..................($19.95)  $19 *
"65816/65802 Assembly Language Programming", Fischer........($21.95)  $20 *
"Programming the 65816", Eyes..............................($22.95)  $21 *
"Apple //e Reference Manual", Apple Computer................($24.95)  $23 *
"Apple //c Reference Manual", Apple Computer................($24.95)  $23 *
"ProDOS Technical Reference Manual", Apple Computer.........($29.95)  $27 *
"Apple ProDOS:  Advanced Features for Programmers", Little..($17.95)  $17 *
"ProDOS Inside and Out", Doms & Weishaar...................($16.95)  $16 *
"DOStalk Scrapbook", Weishaar & Kersey.....................($14.95)  $14 *
"Beneath Apple ProDOS", Worth & Lechner....................($19.95)  $18 *
"Beneath Apple DOS", Worth & Lechner.......................($19.95)  $18 *
"Inside the Apple //c", Little.............................($19.95)  $18 *
"Inside the Apple //e", Little.............................($19.95)  $18 *
"Understanding the Apple //e", Sather......................($24.95)  $23 *
"Understanding the Apple II", Sather.......................($22.95)  $21 *
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95)  $18 *
"Apple ][ Circuit Description", Gayler.....................($22.95)  $21 *
"Assem. Language for Applesoft Programmers", Finley & Myers.($18.95)  $18 *
"Now That You Know Apple Assembly Language...", Gilder......($19.95)  $18 *
"Enhancing Your Apple II, vol. 1", Lancaster...............($15.95)  $15 *
"Enhancing Your Apple II, vol. 2", Lancaster...............($17.95)  $17 *
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95)  $20 *
"Microcomputer Graphics", Myers............................($14.95)  $14 *
"Assembly Lines -- the Book", Wagner.......................($19.95)  $18 *
"AppleVisions", Bishop & Grossberger.......................($39.95)  $36 *

* On these items add $2.00 for the first item and
$.75 for each additional item for US shipping.
+ Inquire for shipping cost.
Foreign customers inquire for postage needed.
Texas residents please add 6 1/8 % sales tax to all orders.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
***                    (214) 324-2050                    ***
*** Master Card, VISA, Discover and American Express ***

list. 'Closing' the file just means updating the catalog sector with the final length and the correct file type. The routine beginning at line 20000 is just a dummy to come up with something to put in the file.

Lines 1420-1460 provide fixed addresses for BASIC to call the various routines. At line 1520, OPEN.USER.FILE uses the DOS CATALOG.INDEX and the catalog sector in memory to look up the location of the first track/sector list, falling into READ.TS.LIST to get that into memory.

ALLOCATE.FILE, beginning at line 1790, is the heart of this program. We start out by remembering the location of the current t/s list and reading the VTOC. We then install our quick patches into DOS. After initializing the offset, program length (1 because of the first t/s list sector) and index, we're ready to start allocating.

We call NEXT.TS.ENTRY to see if there is already another sector allocated. If so it's on to lines 2060-2140 to count that sector and see if we're done. If NEXT.TS.ENTRY returns with .EQ. status then we need another sector and must call ALLOCATE.SECTOR to get it. That routine returns the sector number in the A-register, so we store that and get the track number from the FM's variable at $B5F1. We need to decrement the Y-register because NEXT.TS.ENTRY has already moved TS.INDEX to the next position. Once we have the new sector we count it and, if we're not done, go back for more.

Once we have all the sectors we need, we just call the DOS routine to free up the reserved but unused sectors in the current track, clean up our patches, and finally write out the revised VTOC. Then we write out the current t/s list and we're finished.

READ and WRITE.NEXT.SECTOR just set up the IOB, call NEXT.TS.ENTRY to get a sector location, and call RWTS. If an error occurs it is passed back to BASIC for handling. If NEXT.TS.ENTRY return .EQ. status here, then we have hit the end of the file. This feature isn't used in this demo, but is necessary when we use the same techniques to read a file.

NEXT.TS.ENTRY, at line 2550, starts out by getting the current index into the list. Normally that index is greater than zero, so the routine bumps the index for the next call and returns the next sector and track in Y and A. If the index was zero, then we're finished with that sector of the list and go on to line 2640 to see if there is another sector. If there is another sector then we pick up its location, remember whether we're in the middle of reading or writing, and go get the next piece of the list. Then it's back to the top to get the next entry. Note that READ.TS.LIST resets TS.INDEX to the beginning.

If there isn't another sector we drop down to line 2760. If we're not in the middle of allocating then this means we're at the end of the file, so we return with a zero in the A-register. When we are allocating it means we need to make

another t/s list sector. To do this we first call ALLOCATE.SECTOR to find a place for it, then set the pointer bytes in the current sector, recover the current sector's location, and write it out. Then we remember where this sector is going to go, zero the buffer, update the offset, and count the new sector.

FILE.MANAGER.ERROR just cleans out the DOS patches and passes the error code back to the Applesoft program.` REMOVE and INSTALL.DOS.PATCHES very straightforwardly do just that.

Much of this code is adapted from the S-C Word Processor's method of reading and writing files. My thanks to Bob S-C and Bobby Deen for the inspiration.

```
10  HIMEM: 37888
20 D$ =  CHR$ (4)
30  PRINT D$"BLOAD B.PREALLOCATE,A$9400"
40 PA = 37888
100 TYPE = 0: REM  Change for Non-Text File
110 EC = 3: POKE EC,0: REM Error Code
120 LN = 3: REM File Length, in Sectors
130  POKE 0,LN - LN / 256: POKE 1,LN / 256
140 F$ = "FILENAME"
1000  ONERR  GOTO 10000
1010  PRINT D$"UNLOCK"F$
1020  ONERR  GOTO 11010
1030  CALL PA: REM Locate File
1040  IF  PEEK (EC) THEN 11000
1050  CALL PA + 3: REM Allocate File
1060  IF  PEEK (EC) THEN 11000
1070  PRINT D$"UNLOCK"F$
1080  CALL PA: REM Locate File
1090  IF  PEEK (EC) THEN 11000
1100  FOR I = 1 TO LN
1110  GOSUB 20000: REM Get a Sector
1120  CALL PA + 9: REM Write a Sector
1130  IF  PEEK (EC) THEN 11000
1140  NEXT
1150  PRINT D$"UNLOCK"F$
1160 CL =  PEEK (45980) + 46311
1170  POKE CL, PEEK (4): POKE CL + 1, PEEK (5): REM Catalog File Length
1180  POKE  PEEK (45980) + 46280,TYPE: REM If New, Non-Text File
1190  CALL PA + 12: REM Rewrite Catalog Sector
1200  IF  PEEK (EC) THEN 11000
1210  END
10000  POKE 216,0
10010 ER =  PEEK (222): IF ER <  > 6 THEN 11020
10020  PRINT D$"OPEN"F$: PRINT D$"CLOSE"
10030  GOTO 1020
11000 ER =  PEEK (EC): GOTO 11020
11010 ER =  PEEK (222)
11020  PRINT "ERROR #"ER
11030  POKE 216,0
12000  END
20000  FOR Z = 0 TO 255
20010  POKE 39590,A + 193
20020  NEXT
20030 A = A + 1
20040  RETURN
```

# 7 Meg IIe/1 Meg IIc

**Want the best & easiest-to-use (most support) RAM card on the market — at the best price? Buy Checkmate Technology's State-of-the-Art MULTIRAM RGB RAM CARD™ from us w/BATTERY BACKED-UP RAM options that save programs like AppleWorks for 10+ years!** FAST CHEAP REPLACEMENT FOR HARD DISKS, USER EXPANDABLE TO 7 MEGABYTES, compatible with all (100%) 3rd party software/hardware, optional 65C816 Co-Processor card ($147), sharp 80 columns/Double Hi-Res, & FREE RGB*! It's a direct substitute for Ramworks I-III™ by Applied Engineering (also known as Preferred Computing) or Apple 80 col cards, has a **5 YR WARRANTY, 15 day money back guarantee, & FREE AUTOMATIC SOFTWARE UPDATES ONLY FROM US!**

**MultiRam RGB** expands to 1 Meg RAM + optional 4 Meg's **MultiRam Plus BATTERY PROTECTED RAM.** Optional 2 Meg's STATIC RAM here soon — for a total of 7 megabytes! **MultiRam IIe** expands to 768k. And the **software & manual is the best on the market!**

**FREE APPLEWORKS EXPANDER SOFTWARE WITH EVERY CARD** — loads ALL versions of Appleworks (in 30 seconds w/AW 2.0), even printer routines, runs faster, expands Desktop over 2048k, stores 23,000+ records & 4,800+ word processor lines, auto-segments large files onto multiple disks! **FREE** AW TIME/DAY/DATE ON-SCREEN w/ clock, AUTO-COPY TO RAM, PRODOS/DOS 3.3. RAM DISK & RAM TEST. Optional CP/M & Pascal Ram disks ($20/ea)!

| | MultiRam RGB Card | MultiRam IIe Card |
|---|---|---|
| 64k **MultiRam** | 169. | 129. |
| 320k **MultiRam** | 199. | 164. |
| 576k **MultiRam** | 229. | 199. |
| 832k **MultiRam** | 254. | 234. |
| **1024k MultiRam** | **279.** | ← ← ← ← |

| | MultiRam Plus Piggyback | |
|---|---|---|
| 256k **MultiRam Plus** | 220. | *NEW* |
| 512k **MultiRam Plus** | 269. | *NEW* |
| 768k **MultiRam Plus** | 316. | *NEW* |
| 1024k **MultiRam Plus** | 359. | *NEW* |
| 2-4 meg **MultiRam Plus** | Call | *NEW* |
| Battery/AC Kit (for Plus) | 36. | *NEW* |

| | |
|---|---|
| **Apple IIe Enhancement Kit** | **62.** |
| **Cermetek 1200 Baud internal modem IIe/II+/IIgs** | **179.** |
| **Incomm 2400 Baud external univ. modem** | **319.** |
| **Clockworks Clock (Thunder/Time HO™ compat)** | **89.** |
| **13" RGB Monitor (free cable)** | **322.** |
| **ProAPP 10/20 Meg Hard Disks (+ FREE software)** | **Call** |
| **Apple Compatible Drives IIe/II+ (IIgs/IIc +15)** | **135.** |
| **NETWORKING UP TO 256 APPLE IIc/IIe/IIgs** | **Call** |

Terms: Add $4-Ground or $6-Air shipping & phone # to each U.S. MultiRam card order (foreign orders/FPO/APO extra). Add 3% for MasterCard/Visa (include #/expir) & P.O.'s. For fast delivery send Cashier's/Certified check/Money Order. C.O.D. (add $6) & personal checks accepted (allow 16 days). Tex res add 7 ¼% tax.

**Checkmate Technology's MultiRam CX** card easily expands your IIc to 640k and has a **CX+ Piggyback** to add another 512k **(over 1152k total)!** It's 100% compatible with all IIc software/hardware & upgradable with a 65C816 kit ($105) — all using about 50% less power than Z-RAM! It has the **SAME FREE SOFTWARE/UPDATES & 5 YR WARRANTY AS IIe CARDS.**

| | |
|---|---|
| **256k MULTIRAM CX** | **179.** |
| **512k MULTIRAM CX** | **209.** |
| **768k MULTIRAM CX Combo** | **328.** |
| **1024k MULTIRAM CX Combo** | **354.** |
| **256k MULTIRAM CX+** | **210.** |
| **512k MULTIRAM CX+** | **222.** |

| | |
|---|---|
| **AutoWorks (Best AppleWorks Macro+)** | **37.** |
| **1200 Baud IIc Modem (+$20/cable)** | **159.** |
| **Clip-on IIc Rechargable Battery** | **124.** |
| **IIc Flat Panel Display (C-Vue)** | **299.** |
| **IIc System Clock** | **66.** |

**WHY BUY FROM COIT VALLEY COMPUTERS RATHER THAN SOME MAIL ORDER HOUSES?** Only we offer a **15 day MultiRam complete money back satisfaction guarantee,** extra software, more support, **free automatic software updates,** free 64k w/ each 256k/512k/768k IIe/RGB card. We know the products, & stock them for fast shipping! CALL FOR QUESTIONS, PRICES, DISCOUNTS. **SCHOOLS & GROUPS WELCOME.**

Autoworks, Ramworks III/Timemaster II H.O./Z ram, respective trademarks of Software Touch, Applied Engineering. ***Call for RGB Connectors/cables (not included).**

## ORDER NOW!

**COIT VALLEY COMPUTERS** (214) 234-5047
14055 Waterfall Way Dallas, Texas 75240

NAME ______________________

ADDRESS ______________________

CITY ______________ STATE ________ ZIP ______

PHONE (    ) ______________________

SIGNATURE

| QTY | DESCRIPTION | PRICE |
|---|---|---|
| | | |
| | | |
| | | |

MC/VS ______________ SHIP. ______

______________ EXP ______ TOTAL ______

## COIT VALLEY COMPUTERS
## (214) 234-5047

MCI 2969684
Telex 6502969684 (WUI)

14055 Waterfall Way
Dallas, Texas 75240

```
                   1000 *SAVE S.PREALLOCATE
                   1010 *--------------------------------
00-                1020 COUNT           .EQ 0,1
02-                1030 EOF.FLAG        .EQ 2
03-                1040 ERROR.FLAG      .EQ 3
04-                1050 PGM.LENGTH      .EQ 4,5
06-                1060 TS.INDEX        .EQ 6
07-                1070 TS.OFFSET       .EQ 7,8
09-                1080 THIS.TS.TRACK   .EQ 9
0A-                1090 THIS.TS.SECTOR  .EQ $A
0B-                1100 ALLOCATE.FLAG   .EQ $B
                   1110
03D9-              1120 CALL.RWTS   .EQ $3D9
03E3-              1130 GET.IOB     .EQ $3E3
                   1140
9AA6-              1150 DATA.BUFFER .EQ $9AA6
9BA6-              1160 TS.BUFFER   .EQ $9BA6
                   1170
AFF7-              1180 DOS.READ.VTOC    .EQ $AFF7
AFFB-              1190 DOS.WRITE.VTOC   .EQ $AFFB
B244-              1200 ALLOCATE.SECTOR  .EQ $B244
B2C3-              1210 RELEASE.SECTORS  .EQ $B2C3
B39C-              1220 CATALOG.INDEX    .EQ $B39C
B4C6-              1230 CAT.TRACK        .EQ $B4C6 (+ Index)
B4C7-              1240 CAT.SECTOR       .EQ $B4C7 (   "   )
B5C5-              1250 FM.ERROR         .EQ $B5C5
B5F1-              1260 CURRENT.TRACK    .EQ $B5F1
                   1270
B7E8-              1280 IOB         .EQ $B7E8
B7EB-              1290 IOB.VOLUME  .EQ IOB+3
B7EC-              1300 IOB.TRACK   .EQ IOB+4
B7ED-              1310 IOB.SECTOR  .EQ IOB+5
B7F0-              1320 IOB.BUFFER  .EQ IOB+8,9
B7F4-              1330 IOB.OPCODE  .EQ IOB+12
                   1340 *--------------------------------
                   1350        .OP 65C02
                   1360        .OR $9400
                   1370        .TF B.PREALLOCATE
                   1380 *--------------------------------
                   1390 *
                   1400 *      CALLS FROM BASIC
                   1410 *
9400- 4C 0F 94     1420 CALL.PA     JMP OPEN.USER.FILE
9403- 4C 3F 94     1430 CALL.PA.3   JMP ALLOCATE.FILE
9406- 4C A1 94     1440 CALL.PA.6   JMP READ.NEXT.SECTOR
9409- 4C A4 94     1450 CALL.PA.9   JMP WRITE.NEXT.SECTOR
940C- 4C B8 94     1460 CALL.PA.12  JMP RWTS.CALLER
                   1470 *--------------------------------
                   1480 *
                   1490 *      GET A FILE READY
                   1500 *
                   1510 OPEN.USER.FILE
940F- AE 9C B3     1520        LDX CATALOG.INDEX
9412- BD C6 B4     1530        LDA CAT.TRACK,X     t/s list track
9415- 8D EC B7     1540        STA IOB.TRACK
9418- BD C7 B4     1550        LDA CAT.SECTOR,X     & sector
941B- 8D ED B7     1560        STA IOB.SECTOR
                   1570
                   1580 READ.TS.LIST
941E- A9 01        1590        LDA #1              read opcode
9420- 2C           1600        .HS 2C              skip 2 bytes
                   1610 WRITE.TS.LIST
9421- A9 02        1620        LDA #2              write opcode
9423- 8D F4 B7     1630        STA IOB.OPCODE
9426- A9 A6        1640        LDA #TS.BUFFER      set buffer
9428- 8D F0 B7     1650        STA IOB.BUFFER
942B- A9 9B        1660        LDA /TS.BUFFER
942D- 8D F1 B7     1670        STA IOB.BUFFER+1
9430- 20 B8 94     1680        JSR RWTS.CALLER
9433- A9 0C        1690        LDA #$C             point to first t/s pair
9435- 85 06        1700        STA TS.INDEX
9437- A9 9A        1710        LDA /DATA.BUFFER    restore buffer
9439- 8D F1 B7     1720        STA IOB.BUFFER+1
943C- 64 02        1730        STZ EOF.FLAG
943E- 60           1740        RTS
```

```
                 *--------------------------------
                 *
                 *       PRE-ALLOCATE SECTORS
                 *
                 ALLOCATE.FILE
943F- AE 9C B3          LDX CATALOG.INDEX
9442- BD C6 B4          LDA CAT.TRACK,X    remember where
9445- 85 09             STA THIS.TS.TRACK  we're starting
9447- BD C7 B4          LDA CAT.SECTOR,X
944A- 85 0A             STA THIS.TS.SECTOR
944C- 20 F7 AF          JSR DOS.READ.VTOC
944F- 20 46 95          JSR INSTALL.DOS.PATCHES
9452- 64 07             STZ TS.OFFSET
9454- 64 08             STZ TS.OFFSET+1
9456- A9 01             LDA #1
9458- 85 04             STA PGM.LENGTH     count first t/s list
945A- 64 05             STZ PGM.LENGTH+1
945C- A9 0C             LDA #$C            start of t/s pairs
945E- 85 06             STA TS.INDEX
9460- 85 0B             STA ALLOCATE.FLAG  make non-zero

9462- 20 C6 94   .1     JSR NEXT.TS.ENTRY  (bumps TS.INDEX)
9465- D0 10             BNE .2             .NE. if one already present
9467- 20 44 B2          JSR ALLOCATE.SECTOR
946A- A4 06             LDY TS.INDEX
946C- 88                DEY
946D- 99 A6 9B          STA TS.BUFFER,Y    set sector
9470- AD F1 B5          LDA CURRENT.TRACK
9473- 88                DEY
9474- 99 A6 9B          STA TS.BUFFER,Y    and track

9477- E6 04      .2     INC PGM.LENGTH     count it
9479- D0 02             BNE .3
947B- E6 05             INC PGM.LENGTH+1
947D- C6 00      .3     DEC COUNT
947F- D0 E1             BNE .1             back for more
9481- A5 01             LDA COUNT+1        are we done?
9483- F0 04             BEQ .4
9485- C6 01             DEC COUNT+1        no, keep going
9487- 80 D9             BRA .1

9489- 20 C3 B2   .4     JSR RELEASE.SECTORS    rest of track
948C- 20 43 95          JSR REMOVE.DOS.PATCHES clean house
948F- 20 FB AF          JSR DOS.WRITE.VTOC     update VTOC
9492- 64 0B             STZ ALLOCATE.FLAG      done allocating
9494- A5 09             LDA THIS.TS.TRACK
9496- 8D EC B7          STA IOB.TRACK
9499- A5 0A             LDA THIS.TS.SECTOR
949B- 8D ED B7          STA IOB.SECTOR
949E- 4C 21 94          JMP WRITE.TS.LIST      write this list
                 *--------------------------------
                 *
                 *       READ OR WRITE A SECTOR
                 *
                 READ.NEXT.SECTOR
94A1- A9 01             LDA #1             read
94A3- 2C                .HS 2C             skip 2
                 WRITE.NEXT.SECTOR
94A4- A9 02             LDA #2             write
94A6- 8D F4 B7          STA IOB.OPCODE
94A9- 18                CLC
94AA- 20 C6 94          JSR NEXT.TS.ENTRY  get track/sector
94AD- D0 03             BNE .1             .EQ. means EOF
94AF- E6 02             INC EOF.FLAG       raise the flag
94B1- 60                RTS                and quit

94B2- 8D EC B7   .1     STA IOB.TRACK
94B5- 8C ED B7          STY IOB.SECTOR

                 RWTS.CALLER
94B8- 9C EB B7          STZ IOB.VOLUME
94BB- 20 E3 03          JSR GET.IOB
94BE- 20 D9 03          JSR CALL.RWTS
94C1- 90 02             BCC .1             .CS. if error
94C3- 85 03             STA ERROR.FLAG     pass code back
```

```
94C5- 60          .1      RTS
                  *--------------------------------
                  *
                  *       RETURN NEXT TRACK/SECTOR
                  *
                  NEXT.TS.ENTRY
94C6- A6 06               LDX TS.INDEX
94C8- F0 0B               BEQ .2            done with this list
94CA- E6 06               INC TS.INDEX      bump index
94CC- E6 06               INC TS.INDEX      for next time
94CE- BC A7 9B            LDY TS.BUFFER+1,X
94D1- BD A6 9B            LDA TS.BUFFER,X   return .EQ. if eof
94D4- 60          .1      RTS

94D5- AD A7 9B    .2      LDA TS.BUFFER+1   track of next t/s list
94D8- F0 16               BEQ .3            no next list
94DA- 8D EC B7            STA IOB.TRACK
94DD- AD A8 9B            LDA TS.BUFFER+2   sector
94E0- 8D ED B7            STA IOB.SECTOR
94E3- AD F4 B7            LDA IOB.OPCODE    read or write
94E6- 48                  PHA               stash it
94E7- 20 1E 94            JSR READ.TS.LIST  read next one
94EA- 68                  PLA
94EB- 8D F4 B7            STA IOB.OPCODE    restore
94EE- 80 D6               BRA NEXT.TS.ENTRY and go on

94F0- A5 0B       .3      LDA ALLOCATE.FLAG are we allocating?
94F2- F0 E0               BEQ .1            no, it's eof time
94F4- 20 44 B2            JSR ALLOCATE.SECTOR for next list
94F7- 8D A8 9B            STA TS.BUFFER+2   sector
94FA- AD F1 B5            LDA CURRENT.TRACK
94FD- 8D A7 9B            STA TS.BUFFER+1   and track
9500- A5 09               LDA THIS.TS.TRACK
9502- 8D EC B7            STA IOB.TRACK
9505- A5 0A               LDA THIS.TS.SECTOR
9507- 8D ED B7            STA IOB.SECTOR
950A- 20 21 94            JSR WRITE.TS.LIST write full sector

950D- AD A7 9B            LDA TS.BUFFER+1
9510- 85 09               STA THIS.TS.TRACK keep track
9512- AD A8 9B            LDA TS.BUFFER+2
9515- 85 0A               STA THIS.TS.SECTOR
9517- A2 00               LDX #0
9519- 9E A6 9B    .4      STZ TS.BUFFER,X   clear the buffer
951C- E8                  INX
951D- D0 FA               BNE .4
951F- 18                  CLC
9520- A5 07               LDA TS.OFFSET
9522- 69 7A               ADC #122          update offset
9524- 85 07               STA TS.OFFSET
9526- 8D AB 9B            STA TS.BUFFER+5
9529- A5 08               LDA TS.OFFSET+1
952B- 69 00               ADC #0
952D- 85 08               STA TS.OFFSET+1
952F- 8D AC 9B            STA TS.BUFFER+6
9532- E6 04               INC PGM.LENGTH    count new list sector
9534- D0 02               BNE .5
9536- E6 05               INC PGM.LENGTH+1
9538- 80 8C       .5      BRA NEXT.TS.ENTRY and go on
                  *--------------------------------
                  FILE.MANAGER.ERROR
953A- 20 43 95            JSR REMOVE.DOS.PATCHES clean house
953D- AD C5 B5            LDA FM.ERROR
9540- 85 03               STA ERROR.FLAG          and report error
9542- 60                  RTS
                  *--------------------------------
                  *
                  *   HANDLE TWO PATCHES TO FILE MANAGER
                  *
                  REMOVE.DOS.PATCHES
9543- A2 00               LDX #0
9545- 2C                  .HS 2C            skip two bytes
                  INSTALL.DOS.PATCHES
9546- A2 04               LDX #4
9548- BD 61 95            LDA DOS.PATCHES,X
```

```
954B- 8D FD AF 3250        STA $AFFD
954E- BD 62 95 3260        LDA DOS.PATCHES+1,X
9551- 8D 92 B3 3270        STA $B392
9554- BD 63 95 3280        LDA DOS.PATCHES+2,X
9557- 8D 93 B3 3290        STA $B393
955A- BD 64 95 3300        LDA DOS.PATCHES+3,X
955D- 8D 94 B3 3310        STA $B394
9560- 60       3320        RTS
               3330 *--------------------------------
               3340 DOS.PATCHES
               3350 *   Original Code
9561- AC       3360        .HS AC        LDY opcode
9562- AE 9B B3 3370        LDX $B39B
               3380 *   Patching Code
9565- 60       3390        RTS           don't allow sector allocate
               3400 *                    routine to read/write/VTOC
9566- 4C 3A 95 3410        JMP FILE.MANAGER.ERROR  trap error exits,
               3420 *                    because we didn't really call
               3430 *                    File Manager legitimately.
               3440 *--------------------------------
```

A Smarter Hexadecimal Memory Search.......Bob Sander-Cederlof

Today Rick Hayner called, wanting some help in locating places inside Rak-Ware's DISASM where it calls SETVID and various other subroutines in the monitor. Rick is blind, so he uses the ECHO speech synthesizer to operate his Apple. Programs like DISASM keep un-hooking the speech synthesizer, so he has to patch it until it keeps talking. It takes a long time to search through a whole program for all the possible kinds of code which could un-hook the output device.

We used the monitor "S" command, which can search for a single byte or a two-byte value. This command is not in every Apple monitor, but it was in the one in my computer. We found most of the places, but not all. The trouble was we didn't know exactly what addresses we were looking for, because there are so many possibilities.

Anyway, I kept thinking we needed some kind of searching program which could look through DISASM or whatever more intelligently. What an idea! Let the computer work for us! I came up with SMART.SEARCH, which fills the bill.

SMART.SEARCH can search through a range of memory for any string of bytes up to 255 bytes long. (The listing is prettier if the string length is less than 80, though.) Three strings supplied by the caller define the content of matching strings. The first string is a "mask". Each byte of memory will be ANDed with the corresponding byte in the MASK string before being compared with the key. This lets you ignore certain bits in certain bytes during the search (such as the high bit in ASCII bytes). The key is defined by two strings, providing a lower and an upper limit for values in each byte.

For example, to search for JSR's to subroutines in the monitor, I use a mask string of "FF 00 FF", a lower limit of "20 00 F8", and an upper limit of "20 00 FF". Notice the middle byte always matches, so we trigger on any JSR's to any address between $F800 and $FFFF.

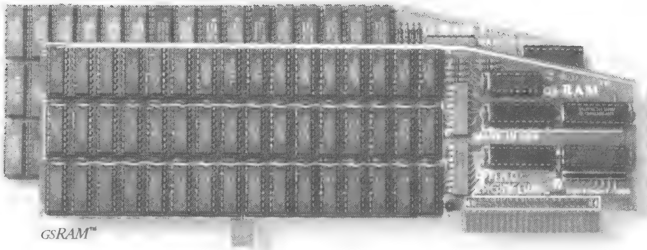
In the listing which follows, Lines 1180-1420 are a demonstration driver for the SMART.SEARCH subroutine. Each call to SMART.SEARCH searches the range from $0800 to $14FF, which happens to be where DISASM loads. I first printed out all the monitor calls, then all the JMPs into the monitor, and then all STA, STX, and STY instructions which store into any address with the low-order byte equal to $36 through $39 (the I/O hooks are at $0036-0039). We might want to also search for JMPs and JSRs to $03EA, which is the DOS I/O hook subroutine, and for stores into the DOS hook area starting at $AA53, but that is "left as an exercise for the reader".

When SMART.SEARCH finds a matching string, it prints out the address of the first byte, and all the bytes of the matching string. SMART.SEARCH displays as many columns of addresses and strings as in can on an 80-column line. If you want a different line length, change the value in line 2050.

# Insist on GSRAM™ When You Buy Your IIGS™

*Expand the IIGS RAM and ROM with the GSRAM or GSRAM Plus with ROM Pak. Available now with 256K to 8 MEG!*

*GSRAM Plus™*



*GSRAM™*

Remember the 16K cards for the II+ and the 64K cards for the IIe? At the time, that much memory seemed like a lot. But when the owners of these memory cards came to us for more memory, many had to throw away their smaller Apple memory cards or try to sell them. Most of our customers told us that had they known about Applied Engineering's larger memory cards when they bought their Apple, they would have purchased them at the same time.

GSRAM and GSRAM Plus are available now, allowing up to 8 MEG of memory expansion. That's 8 times the memory capacity of Apple's card and just look at the benefits that only GSRAM and GSRAM Plus have over Apple's card:

- Lower cost
- Has 6 RAM banks (Apple's card has 4)
- Has memory expansion port
- Has ROM expansion port
- No configuration blocks to set
- No soldered in RAM chips
- Expandable to 8 MEG
- Expands AppleWorks internal limits
- Built-in Hi-Res self-diagnostic software
- 5 year hassle free warranty (Apple has a 90 day warranty)
- Made in USA

### GSRAM for More AppleWorks Power

Only GSRAM and GSRAM Plus eliminate AppleWorks internal memory limits, increasing the maximum number of records available from 6,000 to over 25,000 and only GSRAM and GSRAM Plus increase the number of lines permitted in the word processing mode from 6,000 to over 15,000. And only GSRAM and GSRAM Plus offer a built-in printer buffer so you can continue using Appleworks while your printer is printing. GSRAM and GSRAM Plus even expand the number of lines in the clipboard from 255 to 2047 and will auto segment large files so they can be saved on two or more disks. You can even have Pinpoint or Macroworks and your favorite spelling checker in RAM for instant response. GSRAM and GSRAM Plus will even display the time and date right on the AppleWorks screen. Nothing comes close to enhancing AppleWorks so much.

### Turn Your IIGS into a Giant

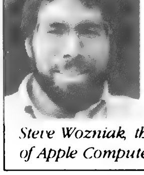
Simply plug GSRAM into the IIGS memory expansion slot and you've got up to *8 megabytes* of RAM at your fingertips—all of it instantly and automatically recognized by the IIGS. GSRAM is compatible with all IIGS software, including AppleWorks, as well as BASIC®, ProDos, DOS 3.3, PASCAL®, "C" and CP/M®.

### Grow by Bytes or Megabytes

We offer GSRAM in two configurations so you can increase your memory 256K at a time (GSRAM) or a megabyte at a time (GSRAM Plus). Both offer full compatibility, lower cost than other boards, and easy expandability. And both are extremely low in power consumption. A fully expanded GSRAM operates at only 375 ma, and GSRAM Plus at only 270 ma (even with 6 megabytes on board!)

### GSRAM—for Normal Memory Requirements

GSRAM is available with 256K, 512K, 1 MEG or 1.5 MEG of memory already on board. If you don't need the full 1.5 MEG now, you can choose a GSRAM with less memory and expand it up to 1.5 MEG in the future—or upgrade to GSRAM Plus for a small charge.



*"In quality, performance, compatibility, expandability and support, Applied Engineering's GSRAM and GSRAM Plus are number one."*

*Steve Wozniak, the creator of Apple Computer*

With an optional piggyback card, you can expand GSRAM even higher than 1.5 MEG! (Other cards are only expandable to 1 MEG.)

### GSRAM Plus—for Growing by Leaps and Bounds

GSRAM Plus is the first Apple memory card to use 1 MEG RAM chips on the main board. It's available with 1 to 6 MEG on board. If you don't need the whole 6 MEG now, you can buy a GSRAM Plus with less memory and easily expand it in the future.

GSRAM Plus can be expanded up to 8 MEG with an optional piggyback card.

### Easy Expansion

Both GSRAM and GSRAM Plus use standard RAM chips that are readily available and just plug right in. So unlike other cards, you'll find expanding your GSRAM or GSRAM Plus easy, convenient and very economical. And with our optional ROM expansion module you can even increase the IIGS's ROM space and all in just one slot.

### Quality and Support of the Industry Leader

Applied Engineering is the largest supplier of Apple peripherals in the world. We *invented* the first large RAM cards for the Apple. With a 5-year "no-hassle" warranty and outstanding technical support, you can be sure GSRAM and GSRAM Plus will deliver the performance you're looking for—or return them within 15 days for a full refund.

| | |
|---|---|
| GSRAM with 256K | $169 |
| GSRAM with 512K | $219 |
| GSRAM with 1 MEG | $299 |
| GSRAM with 1.5 MEG | $379 |
| GSRAM with 2-8 MEG | CALL |
| GSRAM PLUS with 1 MEG | $459 |
| GSRAM PLUS with 2 MEG | $759 |
| GSRAM PLUS with 3-8 MEG | CALL |

### Order today!

See your dealer or call Applied Engineering today, 9 a.m. to 11 p.m. 7 days. Or send check or money order to Applied Engineering. MasterCard, VISA and C.O.D. welcome. Texas residents add 6¼% sales tax. Add $10.00 outside U.S.A.

**APPLIED ENGINEERING™**

*The Apple enhancement experts.*

(214) 241-6060

P.O. Box 798, Carrollton, TX 75006

CP/M is a registered trademark of Digital Research, Inc.

SMART.SEARCH has to be driven by a series of calls, which are assembled together with the SMART.SEARCH subroutine. Before you assemble, change line 1170 to set the origin to a place that does not conflict with the memory you are going to examine. Calls to SMART.SEARCH require a lot of information. I decided to use the technique of including all that information immediately after the JSR SMART.SEARCH. You can see how to code the information by studying lines 1180-1420.

Inside SMART.SEARCH, there are three main sections. The first gets information from the caller and sets up a lot of addresses in page-zero vectors. The second searches for a string that matches. The third prints a matching address and string.

Lines 1450-1740, along with several subroutines in lines 2350-2560, pull in all the caller's information. After the information has all been set up, RETURN contains the address SMART.SEARCH to which should go when it is finished.

Lines 1750-1860 compare for a matching string. Lines 1870-2090 print the matching address and string if one is found. Lines 2110-2230 increment the search address for the next comparison. If not past the end address, this code will branch back to do another comparison.

Lines 2110-2120 and 2250-2330 allow the output to be paused and restarted, or aborted by pressing any key. The RETURN key will terminate the particular SMART.SEARCH in progress.

While writing SMART.SEARCH, I thought of many other possibilities. You might like to try some of them. You could adapt it to search through all the sectors of a floppy disk. You could modify the comparison algorithm to include more options. You could search through all the bytes on an extended memory card (such as RAMWORKS or RAMFACTOR). You could modify the output format. I thought of including a FORMAT string, that could specify both byte order and whether to print the bytes found in hex or in ASCII. Or, you could just use the "shell" which handles the parameter passage, and code an entirely different function. If you have an Apple IIgs, you might want to modify SMART.SEARCH so that it looks like a "Tool", and have it installed at boot-up time.

If you come up with some great ones, let us have a look. We might like to pass them on to the rest of you!

```
                 1000 *SAVE SMART.SEARCH
                 1010 *--------------------------------
00-              1020 PNTR   .EQ $00,01
02-              1030 PEND   .EQ $02,03
04-              1040 PMASK  .EQ $04,05
06-              1050 PLOWER .EQ $06,07
```

```
08-                PUPPER .EQ $08,09
0A-                LENGTH .EQ $0A
0B-                FWIDTH .EQ $0B
0C-                COLUMN .EQ $0C
0D-                RETURN .EQ $0D,0E
                   *--------------------------------
F941-              PRNTAX .EQ $F941
FDED-              COUT   .EQ $FDED
FD8E-              CROUT  .EQ $FD8E
FDDA-              PRBYTE .EQ $FDDA
                   *--------------------------------
                          .OR $3000
                   T
3000- 20 31 30            JSR SMART.SEARCH
3003- 00 08               .DA $0800     Starting Address
3005- 00 15               .DA $1500     Ending address + 1
3007- 03                  .HS 03        Length of strings
3008- FF 00 FF            .HS FF.00.FF Mask string
300B- 20 00 F8            .HS 20.00.F8 Lower limit string
300E- 20 00 FF            .HS 20.00.FF Upper limit string
                   *--------------------------------
3011- 20 31 30            JSR SMART.SEARCH
3014- 00 08               .DA $0800     Starting Address
3016- 00 15               .DA $1500     Ending address + 1
3018- 03                  .HS 03        Length of strings
3019- FF 00 FF            .HS FF.00.FF Mask string
301C- 4C 00 F8            .HS 4C.00.F8 Lower limit string
301F- 4C 00 FF            .HS 4C.00.FF Upper limit string
                   *--------------------------------
3022- 20 31 30            JSR SMART.SEARCH
3025- 00 08               .DA $0800     Starting Address
3027- 00 15               .DA $1500     Ending address + 1
3029- 02                  .HS 02        Length of strings
302A- FF FF               .HS FF.FF     Mask string
302C- 81 36               .HS 81.36     Lower limit string
302E- 9D 39               .HS 9D.39     Upper limit string
3030- 60                  RTS
                   *--------------------------------
                   SMART.SEARCH
3031- 20 8E FD            JSR CROUT
3034- 68                  PLA           GET RETURN ADDRESS
3035- 85 0D               STA RETURN
3037- 68                  PLA
3038- 85 0E               STA RETURN+1
                   *---Get Parameters after JSR-----
303A- 20 E5 30            JSR GET.WORD       Get Starting Address
303D- 86 00               STX PNTR
303F- 85 01               STA PNTR+1
3041- 20 E5 30            JSR GET.WORD       Get Ending Address + 1
3044- 86 02               STX PEND
3046- 85 03               STA PEND+1
3048- 20 E9 30            JSR GET.BYTE       Get Search String Length
304B- 85 0A               STA LENGTH
304D- 0A                  ASL                Calc Field Width
304E- 69 07               ADC #7
3050- 85 0B               STA FWIDTH
3052- 85 0C               STA COLUMN         Start column counter
3054- 20 E9 30            JSR GET.BYTE       Point at Mask String
3057- A5 0D               LDA RETURN
3059- 85 04               STA PMASK
305B- A5 0E               LDA RETURN+1
305D- 85 05               STA PMASK+1
305F- 20 F4 30            JSR ADD.LENGTH.TO.RETURN
3062- 86 06               STX PLOWER         Point at Lower Limit String
3064- 85 07               STA PLOWER+1
3066- 20 F4 30            JSR ADD.LENGTH.TO.RETURN
3069- 86 08               STX PUPPER         Point at Upper Limit String
306B- 85 09               STA PUPPER+1
306D- 20 F4 30            JSR ADD.LENGTH.TO.RETURN
                   *---Compare strings--------------
3070- A0 00        .1     LDY #0
3072- B1 00        .2     LDA (PNTR),Y
3074- 31 04               AND (PMASK),Y
3076- D1 06               CMP (PLOWER),Y
3078- 90 3F               BCC .6
```

```
307A- D1 08     1810        CMP (PUPPER),Y
307C- 90 02     1820        BCC .3        ...YES
307E- D0 39     1830        BNE .6
3080- C8        1840 .3     INY
3081- C4 0A     1850        CPY LENGTH
3083- 90 ED     1860        BCC .2        ...MORE TO COMPARE
                1870 *---Found a matching string------
3085- A5 01     1880        LDA PNTR+1    Print the Address
3087- A6 00     1890        LDX PNTR
3089- 20 41 F9  1900        JSR PRNTAX
308C- A9 AD     1910        LDA #"-"
308E- 20 ED FD  1920        JSR COUT
3091- A0 00     1930        LDY #0        Print the String
3093- B1 00     1940 .4     LDA (PNTR),Y
3095- 20 DA FD  1950        JSR PRBYTE
3098- C8        1960        INY
3099- C4 0A     1970        CPY LENGTH
309B- 90 F6     1980        BCC .4
309D- A9 A0     1990        LDA #" "      Tab to next column
309F- 20 ED FD  2000        JSR COUT
30A2- 20 ED FD  2010        JSR COUT
                2020 *---Check if end of line---------
30A5- A5 0B     2030        LDA FWIDTH
30A7- 65 0C     2040        ADC COLUMN
30A9- C9 50     2050        CMP #80
30AB- 90 05     2060        BCC .5
30AD- 20 8E FD  2070        JSR CROUT
30B0- A5 0B     2080        LDA FWIDTH
30B2- 85 0C     2090 .5     STA COLUMN
                2100 *---Check for pause/abort--------
30B4- AD 00 C0  2110        LDA $C000
30B7- 30 16     2120        BMI .7        PAUSE OR ABORT
                2130 *---Advance Pointer--------------
30B9- E6 00     2140 .6     INC PNTR
30BB- D0 B3     2150        BNE .1
30BD- E6 01     2160        INC PNTR+1
30BF- A5 00     2170        LDA PNTR
30C1- C5 02     2180        CMP PEND
30C3- A5 01     2190        LDA PNTR+1
30C5- E5 03     2200        SBC PEND+1
30C7- 90 A7     2210        BCC .1
30C9- 20 8E FD  2220        JSR CROUT
30CC- 6C 0D 00  2230        JMP (RETURN)
                2240 *--------------------------------
30CF- 8D 10 C0  2250 .7     STA $C010
30D2- C9 8D     2260        CMP #$8D
30D4- F0 0C     2270        BEQ .9        ...ABORT
30D6- AD 00 C0  2280 .8     LDA $C000
30D9- 10 FB     2290        BPL .8        ...PAUSE
30DB- 8D 10 C0  2300        STA $C010
30DE- C9 8D     2310        CMP #$8D
30E0- D0 D7     2320        BNE .6        ...END OF PAUSE
30E2- 6C 0D 00  2330 .9     JMP (RETURN)
                2340 *--------------------------------
                2350 GET.WORD
30E5- 20 E9 30  2360        JSR GET.BYTE
30E8- AA        2370        TAX
                2380 GET.BYTE
30E9- E6 0D     2390        INC RETURN
30EB- D0 02     2400        BNE .1
30ED- E6 0E     2410        INC RETURN+1
30EF- A0 00     2420 .1     LDY #0
30F1- B1 0D     2430        LDA (RETURN),Y
30F3- 60        2440        RTS
                2450 *--------------------------------
                2460 ADD.LENGTH.TO.RETURN
30F4- 18        2470        CLC
30F5- A5 0D     2480        LDA RETURN
30F7- 65 0A     2490        ADC LENGTH
30F9- 85 0D     2500        STA RETURN
30FB- AA        2510        TAX
30FC- A5 0E     2520        LDA RETURN+1
30FE- 69 00     2530        ADC #0
3100- 85 0E     2540        STA RETURN+1
3102- 60        2550        RTS
```

## ProVIEW....................................Bob Sander-Cederlof

For the last year or so Doug McComsey has been fine tuning this program, and the result is a tool which I like, use, and recommend.

ProVIEW is essentially a slick ZAP program for ProDOS disks and memory. On an Apple //c, //e, or //gs in 80-column mode ProVIEW gives you a window into RAM, ROM, ProDOS files, and ProDOS disk blocks. You can examine, modify, and update any portion of a file or disk block. ProVIEW does not recognize DOS disks, and will not work in a II, IIPlus, IIclone, or anything in 40-column mode. Nevertheless, these restrictions are more than overcome by its features and ease-of-use.

The various control and display screens of ProVIEW look and function a lot like Appleworks. As a result, you very quickly outgrow the need for a reference manual. All of the keystrokes you need to use are natural and easy to remember; help is only a keystroke away at all times, using the Apple-? combination.

ProVIEW executes without disturbing the S-C Macro Assembler or BASIC.SYSTEM. When you want to use it, you simply insert the disk and type "-PV". Or, if you have it on your hard disk or RAM-disk you can get it even quicker. When you are through, a simple ESCAPE keystroke gets you back to the assembler or to Applesoft.

Doug has asked us to handle sales of ProVIEW. Our introductory price is only $20. You will get documentation and a copyable disk, with one of the handiest tools you've ever purchased.

## Display the IIgs Tool Tables.............Bob Sander-Cederlof

While exploring in my IIgs, a map is handy. There is a lot of RAM and a lot of ROM, and it is practically impossible to keep everything in my head. It is also ridiculous to try drawing such a map with pencil and paper. Why not use the computer to document itself?

The tools in the IIgs are all accessed through a hierarchy of tables. All you need to know is the address of the first (top) table. The top table starts with a 4-byte value which is the number of secondary tables. Following this count are a list of 4-byte addresses which point to each of the secondary tables. Each secondary table begins with a 4-byte count of the number of tools in that toolset, and continues with a list of 4-byte addresses pointing to the code for each tool.

These tables may be either in RAM or ROM. When you turn on your IIgs certain values are loaded into bank $E1. Among these is a pointer at $E1/03C0 which contains a 3-byte address for the top tool table. In my IIgs this is at $FE/013F. All of the secondary tables are also in bank $FE of ROM. However, after booting the system disk most of the tables have been rebuilt in RAM. About half of the toolsets are not resident in ROM anyway, so those tables are meaningless until the system disk has been booted and the RAM-based toolsets loaded. Many of the ROM-based toolsets are already somewhat obselete, because the system disk builds RAM-based tables and loads modified tool code into RAM.

The following program will start at $E1/03C0 and follow the hierarchy of tables. The addresses for all of the tools in every toolset will be displayed. If your printer is on and connected, the tables will be listed on the printer as well.

Although my program runs entirely in Emulation Mode, it still uses some of the new addressing modes of the 65816. Line 1170 shows an example of long indexed mode, and is part of a loop copying the address of the top table into a page-zero pointer. Line 1390 is one of many examples using the long-indirect-indexed mode. In this mode a 3-byte address in page zero is added to the current value in the Y-register to get the effective address.

In the S-C Macro Assembler this long form is distinguished from the normal 2-byte indirect-indexed form by using a ">" before the operand: ">(zp),Y" for long-indirect-indexed, and "(zp),Y" for regular indirect-indexed. The "greater than" sign indicates mnemonically that the address at "zp" is three bytes long rather than two. The assemblers which followed the lead of ORCA/M use a different syntax: "[zp],Y" for long-indirect-indexed. I chose not to use the square brackets because the earlier Apple keyboards could not generate a left bracket ([), and because the right bracket was already in use for macro parameters. Future versions of S-C Macro may allow the ORCA syntax as well as the current form.

If you have a //gs, you will benefit by entering this program and running it. Once you know where the tools are located, you can dig in with the disassembler and find out what makes them tick!

```
                                .OP 65816
                        *--------------------------------
                        *  This program runs in Emulation Mode
                        *--------------------------------
FD8E-                   CROUT  .EQ $FD8E
FDDA-                   PRBYTE .EQ $FDDA
FDED-                   COUT   .EQ $FDED
                        *--------------------------------
00-                     PTRA        .EQ $00,01,02
03-                     PTRB        .EQ $03,04,05
06-                     TOOL.NUMBER .EQ $06
07-                     TOOLSET.NO  .EQ $07
                        *--------------------------------
                        LIST.TOOL.TABLES
000800- A2 02                  LDX #2
000802- BF C0 03 E1     .1     LDA $E103C0,X      Start with main table
000806- 95 00                  STA PTRA,X         Address of table of toolset addresses
000808- 95 03                  STA PTRB,X
00080A- CA                     DEX                3 bytes per address
00080B- 10 F5                  BPL .1
                        *---Display main table------------
00080D- A0 18                  LDY #Q.TOOLSET          Title for main table
00080F- 20 89 08               JSR QTO
000812- 20 47 08               JSR DISPLAY.A.TABLE     Addresses of toolset tables
                        *---Loop for each toolset--------
000815- A9 01                  LDA #1             Start with toolset #1
000817- 85 07                  STA TOOLSET.NO
000819- A0 3A           .2     LDY #Q.TOOL        Toolset title
00081B- 20 89 08               JSR QTO
00081E- A5 07                  LDA TOOLSET.NO     Print the # too
000820- 20 DA FD               JSR PRBYTE
000823- A0 28                  LDY #Q.LINE        Underline it
000825- 20 89 08               JSR QTO
000828- A5 07                  LDA TOOLSET.NO     Make toolset# into index
00082A- 0A                     ASL                by multiplying by 4
00082B- 0A                     ASL
00082C- A8                     TAY
00082D- B7 03                  LDA >(PTRB),Y      Get address of toolset table
00082F- 85 00                  STA PTRA           lo-byte
000831- C8                     INY
000832- B7 03                  LDA >(PTRB),Y
000834- 85 01                  STA PTRA+1         mid-byte
000836- C8                     INY
000837- B7 03                  LDA >(PTRB),Y
000839- 85 02                  STA PTRA+2         hi-byte
00083B- 20 47 08               JSR DISPLAY.A.TABLE
00083E- E6 07                  INC TOOLSET.NO     Next toolset...
000840- A5 07                  LDA TOOLSET.NO
000842- C7 03                  CMP >(PTRB)        Compare to number of existing sets
000844- 90 D3                  BCC .2             ...more toolsets
000846- 60                     RTS
                        *--------------------------------
                        DISPLAY.A.TABLE
000847- A0 00                  LDY #Q.TITLE       Title for the table
000849- 20 89 08               JSR QTO
00084C- A9 01                  LDA #1             Start with tool #1 in set
00084E- 85 06                  STA TOOL.NUMBER
000850- A0 10           .1     LDY #Q.CRD         Newline and a dollar sign
000852- 20 89 08               JSR QTO
000855- A5 06                  LDA TOOL.NUMBER    Print the tool (function) # in hex
000857- 20 DA FD               JSR PRBYTE
00085A- A0 13                  LDY #Q.DOTS        Connect with dots to address column
00085C- 20 89 08               JSR QTO
00085F- A5 06                  LDA TOOL.NUMBER    Make tool# into an index by
000861- 0A                     ASL                   multiplying by four
000862- 0A                     ASL
000863- A8                     TAY
000864- C8                     INY                Point to the hi-byte
000865- C8                     INY
000866- B7 00                  LDA >(PTRA),Y
000868- 20 DA FD               JSR PRBYTE         Print hi-byte
00086B- A9 AF                  LDA #"/"           Using bb/hhll format
00086D- 20 ED FD               JSR COUT
000870- 88                     DEY                Print mid-byte
000871- B7 00                  LDA >(PTRA),Y
000873- 20 DA FD               JSR PRBYTE
000876- 88                     DEY                Print lo-byte
000877- B7 00                  LDA >(PTRA),Y
000879- 20 DA FD               JSR PRBYTE
00087C- E6 06                  INC TOOL.NUMBER    Next tool # in set
00087E- A5 06                  LDA TOOL.NUMBER
000880- C7 00                  CMP >(PTRA)        Out of the set yet?
000882- 90 CC                  BCC .1             ...No, still more tools
000884- 60                     RTS
```

```
                          1860 *--------------------------------
000885- 20 ED FD          1870 QTO1     JSR COUT         Print a string
000888- C8                1880          INY
000889- B9 8F 08          1890 QTO      LDA QTS,Y        Enter here with Y indexing message
00088C- D0 F7             1900          BNE QTO1
00088E- 60                1910          RTS
                          1920 *--------------------------------
                          1930 QTS
00-                       1940 Q.TITLE      .EQ *-QTS
00088F- 8D                1950          .HS 8D
000890- D4 EF EF EC
000894- A0 A0 A0 C1
000898- E4 E4 F2 E5
00089C- F3 F3             1960          .AS -/Tool   Address/
00089E- 00                1970          .HS 00
10-                       1980 Q.CRD    .EQ *-QTS
00089F- 8D A4 00          1990          .HS 8D.A4.00
13-                       2000 Q.DOTS   .EQ *-QTS
0008A2- AE AE AE A4       2010          .AS -/...$/
0008A6- 00                2020          .HS 00
18-                       2030 Q.TOOLSET    .EQ *-QTS
0008A7- 8D                2040          .HS 8D
0008A8- A0 D4 EF EF
0008AC- EC F3 E5 F4
0008B0- A0 D4 E1 E2
0008B4- EC E5 F3          2050          .AS -/ Toolset Tables/
28-                       2060 Q.LINE   .EQ *-QTS
0008B7- 8D                2070          .HS 8D
0008B8- AD AD AD AD
0008BC- AD AD AD AD
0008C0- AD AD AD AD
0008C4- AD AD AD AD       2080          .AS -/----------------/
0008C8- 00                2090          .HS 00
3A-                       2100 Q.TOOL       .EQ *-QTS
0008C9- 8D 8D 8D          2110          .HS 8D.8D.8D
0008CC- D4 EF EF EC
0008D0- F3 A0 E9 EE
0008D4- A0 D3 E5 F4
0008D8- A0 A4             2120          .AS -/Tools in Set $/
0008DA- 00                2130          .HS 00
```

Display the 65802 Registers.....................Bob Boughner

Shortly after I installed the 65802 chip in my Apple IIPlus, I inadvertently tried to execute a BRK instruction while in the 16-bit Native Mode. Needless to say, things did not work too well. After digging into the problem, I uncovered two areas where difficulties occur. The first is that the monitor ROMs implicitly assume 8-bit word lengths; going into them while in the 16-bit mode will definitly not work.

The second area, and even more serious, is that the BRK vector is located at a different place when you are in Native Mode. In Emulation Mode, or in a normal 6502 or 65C02, the BRK vector is combined with the IRQ vector and is at $FFFE and $FFFF. When you are in Native Mode in a 65802 or 65816, the BRK vector is separated from the IRQ vector. The Native Mode IRQ vector is at $FFEE and $FFEF, while the Native Mode BRK vector is at $FFE6 and $FFE7. A separate vector is required in Native Mode because there is no B-bit in the Native-mode status register.

After this experience, I decided to write a subroutine that would display all the registers, regardless of what mode I was in. The program which follows is the result of that effort as modified by Bob S-C. The program will only work in a 65802 or 65816; if your Apple has a 65C02 or 6502, you are out of luck because I used some instructions and addressing modes not in those processors. As you will notice, I decided not to bother with displaying the program bank and data bank registers, because in a 65802 these have no effect on anything. I also assume the program will be running in an Apple II machine below the IIgs level, so these registers will probably always be zero.

I also assumed in my code that the stack always stays in page one. This may not be a good assumption, but it simplifies a lot. Whenever a program switches to emulation mode, the high byte of the S-register gets changed to 01. Since I am using monitor subroutines, I have to use emulation mode. I further have assumed that the stack-relative addressing mode will work. This means that the stack cannot wrap around during the call and execution of DISPLAY.REGISTERS. If it does, the values printed out may not be correct.

I decided to split the display so that it will fit on a 40-column screen. The PC address and the A-, X-, and Y-registers display on the first line. The D- and S-registers display on the second line, followed by the nine status bits. I break out the status bits and display the bit name in normal video if the bit is 0. If a status bit is 1, its name will display in INVERSE on a //e or //c 80-column screen, and in FLASHING on a 40-column screen. By changing line 1650 to "AND #$DF", and line 1990 to "FLAGS .AS -/eczidxmvn/", you can make 0 status bits display in lower case and 1 status bits display in upper case (assuming your Apple has lower-case display).

Bob S-C made one change which saved a lot of code but will greatly offend some programmers. He added some self-modifying code! Line 1480 stores a value into the instruction in line

1490, building a "LDA ...,S" instruction with the proper offset to pick up a byte out of the stack.

Lines 1190-1220 are the standard way of entering a program which can be called from both Native or Emulation Mode. The first PHP saves the actual status register, and the second one saves the value that was in the E-bit. The CLC-XCE gets us into Native Mode, so that we can save the registers. Since we might have been in full 16-bit native mode, we have to be in that mode when we save the A-, X-, and Y-registers.

Lines 1230-1270 save the registers by pushing them on the stack. Lines 1290 set the D-register to 0000, so that we can call monitor subroutines later. Lines 1320-1350 form a modified return address, so that we can print out the actual address of the JSR DISPLAY.REGISTERS instruction. This really is not that necessary, but I liked it better this way. Lines 1370-1400 push the address on the stack of the stack pointer before the JSR DISPLAY.REGISTERS was executed. Now there are quite a few bytes on the stack, so I will draw a map. The numbers indicate the value that can be used in a "offset,S" addressing mode to access the bytes.

```
        $10,S  return-hi
        $0F,S    "   -lo
        $0E,S  P-register (status)
        $0D,S      "      (E-bit)
        $0C,S  D-register-hi
        $0B,S      "     -lo
        $0A,S  Y-register-hi
        $09,S      "     -lo
        $08,S  X-register-hi
        $07,S      "     -lo
        $06,S  A-register-hi
        $05,S      "     -lo
        $04,S  call addrs-hi
        $03,S      "     -lo
        $02,S  stack addr-hi
        $01,S      "     -lo
```

Lines 1420-1550 display the register contents of the 16-bit registers. A format string controls the loop. The bytes in the string which are ASCII characters with the high bit set are printed just as they are. Bytes in the format string which are hex values below $20 are offsets into the stack. These are used to pick up bytes out of the stack for printing in hexadecimal.

Lines 1560-1690 display the nine status bits. I first shove the E-bit value into CARRY, then pick up the rest of the bits in the A-register. The loop then shifts A and CARRY, so that I can effectively squeeze nine bits into the eight-bit A-register.

Lines 1710-1830 restore all the registers and return to the caller in the same mode and with the same status as at the time of call.

I added a little test routine to the end, lines 2010-2100. This loads a few test values into registers in native mode and displays them. The result looked like this:

```
0898-  A=1234  X=5678  Y=ABCD
       D=0000  S=01E2  NVMXDIZCE
```

with N and C in inverse or flashing mode.

When I called DISPLAY.REGISTERS directly by typing $800G from within the ProDOS version of the S-C Macro Assembler, the display looked like this:

```
8B52-  A-00DC  X=0002  Y=0000
       D=0000  S=01E2  NVMXDIZCE
```

with M, X, and E in inverse or flashing mode.

I use my DISPLAY.REGISTERS subroutine when I am debugging a new program, by assembling it in and calling it at strategic points. I am working on a more complete DEBUG program which will allow STEP and TRACE for all of the 65802 opcodes. I am using Bob S-C's DISASM.816 (published in AAL in March 1985 and March 1986), and Steve Wozniak's 6502 TRACE from the old monitor ROM.

```
                         1000 *SAVE S.REG.DISP
                         1010 *--------------------------------
                         1020        .OP 65802
                         1030 *--------------------------------
FDDA-                    1040 MON.PRBYTE .EQ $FDDA
FDED-                    1050 MON.COUT   .EQ $FDED
                         1060 *--------------------------------
                         1070 *   SUBROUTINE TO DISPLAY 65802 REGISTERS
                         1080 *   By Bob Boughner and Bob S-C
                         1090 *
                         1100 *   I assume this routine will be used in
                         1110 *   an Apple with only 64K address space,
                         1120 *   so PBR and DBR are always 00.
                         1130 *
                         1140 *   I also assume that the S-register is
                         1150 *   always in page 1, because it must be
                         1160 *   there for monitor subroutines to work.
                         1170 *--------------------------------
                         1180 DISPLAY.REGISTERS
000800- 08               1190        PHP          SAVE CALLER'S STATUS
000801- 18               1200        CLC          ENTER NATIVE MODE
000802- FB               1210        XCE
000803- 08               1220        PHP          SAVE CALLER'S E-BIT
000804- C2 38            1230        REP #$38     CLR M,X,D (16-BIT, NOT DECIMAL MODE)
000806- 0B               1240        PHD          SAVE D-REGISTER
000807- 5A               1250        PHY               Y-REGISTER
000808- DA               1260        PHX               X-REGISTER
000809- 48               1270        PHA               A-REGISTER
                         1280 *--------------------------------
00080A- A9 00 00         1290        LDA ##0000   D=0000, so no problems with use of monitor
00080D- 5B               1300        TCD          subroutines.
                         1310 *--------------------------------
00080E- A3 0B            1320        LDA 11,S     Get and adjust Return Address to point to
000810- 3A               1330        DEC          the JSR that got us in here.
000811- 3A               1340        DEC
000812- 48               1350        PHA
                         1360 *--------------------------------
000813- 3B               1370        TSC          Get and adjust Stack Pointer
000814- 38               1380        SEC
000815- E9 0D 00         1390        SBC ##0013
000818- 48               1400        PHA
                         1410 *--------------------------------
000819- 38               1420        SEC          Get into emulation mode for display
00081A- FB               1430        XCE
                         1440 *---Display 16-bit registers-----
00081B- A0 00            1450        LDY #0
00081D- B9 59 08         1460 .1     LDA FORMAT,Y
000820- 30 0A            1470        BMI .3
000822- 8D 26 08         1480        STA .2+1      *** SELF-MODIFYING CODE ***
000825- A3 00            1490 .2     LDA *-*,S
```

```
000827- 20 DA FD    1500        JSR MON.PRBYTE
00082A- 80 03       1510        BRA .4
00082C- 20 ED FD    1520 .3     JSR MON.COUT
00082F- C8          1530 .4     INY
000830- C0 29       1540        CPY #FORMAT.LEN
000832- 90 E9       1550        BCC .1
                    1560 *---Display status bits----------
000834- A0 08       1570        LDY #8       9 bits altogether
000836- A3 0D       1580        LDA 13,S     Get E-bit into CARRY
000838- 4A          1590        LSR
000839- A3 0E       1600        LDA 14,S     Get the rest of the bits
00083B- 2A          1610 .5     ROL          First time this puts E into bit0
00083C- 48          1620        PHA          Save status byte
00083D- B9 82 08    1630        LDA FLAGS,Y  Get bit name
000840- 90 02       1640        BCC .6       Print NORMAL if bit=0
000842- 29 7F       1650        AND #$7F     Print INVERSE or FLASH if bit=1
000844- 20 ED FD    1660 .6     JSR MON.COUT
000847- 68          1670        PLA          Get status byte again
000848- 88          1680        DEY          Next bit
000849- 10 F0       1690        BPL .5
                    1700 *---Restore everything & return--
00084B- 18          1710        CLC          NATIVE MODE
00084C- FB          1720        XCE
00084D- C2 30       1730        REP #$30
00084F- 68          1740        PLA          POP OFF S-REG VALUE
000850- 68          1750        PLA          POP OFF PC VALUE
000851- 68          1760        PLA          RESTORE A-REG
000852- FA          1770        PLX                  X-REG
000853- 7A          1780        PLY                  Y-REG
000854- 2B          1790        PLD                  D-REG
000855- 28          1800        PLP          E-BIT
000856- FB          1810        XCE
000857- 28          1820        PLP          STATUS
000858- 60          1830        RTS
                    1840 *--------------------------------
000859- 04 03       1850 FORMAT .HS 04.03        PC of JSR DISPLAY.REGISTERS
00085B- AD A0 A0 C1
00085F- BD          1860        .AS -/-  A=/
000860- 06 05       1870        .HS 06.05        A-register
000862- A0 A0 D8 BD 1880        .AS -/  X=/
000866- 08 07       1890        .HS 08.07        X-register
000868- A0 A0 D9 BD 1900        .AS -/  Y=/
00086C- 0A 09 8D    1910        .HS 0A.09.8D     Y-register, RETURN
00086F- A0 A0 A0 A0
000873- A0 A0 A0 C4
000877- BD          1920        .AS -/       D=/
000878- 0C 0B       1930        .HS 0C.0B        D-register
00087A- A0 A0 D3 BD 1940        .AS -/  S=/
00087E- 02 01       1950        .HS 02.01        S-register
000880- A0 A0       1960        .AS -/  /
29-                 1970 FORMAT.LEN .EQ *-FORMAT
                    1980 *--------------------------------
000882- C5 C3 DA C9
000886- C4 D8 CD D6
00088A- CE          1990 FLAGS  .AS -/ECZIDXMVN/
                    2000 *--------------------------------
00088B- 18          2010 T      CLC
00088C- FB          2020        XCE
00088D- C2 30       2030        REP #$30
00088F- A9 34 12    2040        LDA ##$1234
000892- A2 78 56    2050        LDX ##$5678
000895- A0 CD AB    2060        LDY ##$ABCD
000898- 20 00 08    2070        JSR DISPLAY.REGISTERS
00089B- 38          2080        SEC
00089C- FB          2090        XCE
00089D- 60          2100        RTS
                    2110 *--------------------------------
```

## ProDOS-based Intelligent Disassembler.....Bob Sander-Cederlof

In response to the requests of many of you, I have at long-last developed a disassembler which runs under ProDOS. (This new product is distinctly different from the Rak-Ware DISASM, which runs under DOS 3.3. The Rak-Ware product is still the best one to use if you are using DOS.) Here are some of the features of the new S-C ProDOS DISASM:

* Input is from one or more binary object files, including file types BIN and SYS.

* Output is to one or more "S-C" type (compressed source code) files.

* Generates comment lines before each label listing all references to that label.

* Disassembly is "script" driven, allowing incremental enhancement.

* Input files may be positioned to specific starting addresses.

* Decodes ProDOS "MLI" calls as such.

* Allows pre-named symbols up to 32-characters long.

Of all the features, the most important may be the "script". This is essentially a "program", written in "disassembly language". The script allows you to define which input files to include and which output files to generate, to name symbols such as monitor entry points and major subroutines in your program being disassembled, to define table areas, and even to insert comments.

The script itself is written using the standard S-C Macro Assembler, and may be saved on a source file just as an assembly-language program would. As you gain knowledge about the program you are disassembling, you can add lines to the script.

Version 1.0 handles all of the 65C02 instruction set. Future enhancements which which are definitely planned include expanding to include the entire 65816 instruction set. Version 1.0 is for sale now for $50 including the commented source code.

Apple Assembly Line (ISSN 0889-4302) is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are $1.80 each (other countries inquire for postage). A subscription to the newsletter and the Monthly Disk containing all source code is $64 per year in the US, Canada and Mexico, and $87 to other countries.